SUBMITTED TO: FOURTH IFIP WORKING CONF. ON DEPENDABLE COMPUTING FOR CRITICAL APPLICATIONS JAN 4-6, 1994 SAN DIEGO, CA



# Optimal Message Log Reclamation for Uncoordinated Checkpointing



*Yi-Min Wang and W. Kent Fuchs*

Primary contact: W. Kent Fuchs

Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
University of Illinois
1308 West Main Street
Urbana, IL 61801

E-mail: fuchs@crhc.uiuc.edu
Phone: (217) 333-9731
FAX: (217) 244-5686





## Abstract

Uncoordinated checkpointing for message-passing systems allows maximum process autonomy and general nondeterministic execution, but suffers from potential domino effect and the large space overhead for maintaining checkpoints and message logs. Traditionally, it has been assumed that only *obsolete* checkpoints and message logs before the global recovery line can be garbage-collected. Recently, an approach to identifying all garbage checkpoints based on recovery line transformation and decomposition has been developed. We show in this paper that the same approach can be applied to the problem of identifying all garbage message logs for systems requiring message logging to record in-transit messages. Communication trace-driven simulation for several parallel programs is used to evaluate the proposed algorithm.



*Key words: checkpointing, rollback recovery, message-passing systems, message logging, garbage collection*



[1]This research was supported in part by the National Aeronautics and Space Administration (NASA) under Grant NASA NAG 1-613, in cooperation with the Illinois Computer Laboratory for Aerospace Systems and Software (ICLASS), and in part by the Department of the Navy and managed by the Office of the Chief of Naval Research under Contract N00014-91-J-1283

# 1 Introduction

Checkpointing and rollback recovery is an effective approach to recovering from both hardware and software errors. During normal execution, the state of each process is periodically saved as a *checkpoint* on stable storage, and can be restored after a failure in order to avoid the costly reexecution from the very beginning. Numerous checkpointing and recovery techniques for message-passing systems have been proposed in the literature. They can be classified into three primary categories: uncoordinated checkpointing, coordinated checkpointing and log-based approach. *Uncoordinated checkpointing* [1–3] allows maximum process autonomy and general nondeterministic execution. Each process takes its checkpoints independently and keeps track of the dependencies among checkpoints resulted from message communications. When a failure occurs, the dependency information is used to determine the *recovery line* to which the system should roll back. The major disadvantages of uncoordinated checkpointing have been the potential *domino effect* [4, 5], i.e., when *cyclic rollback propagation* prevents recovery line progression, and the space overhead for maintaining checkpoints and message logs.

*Coordinated checkpointing* [6–11] eliminates the domino effect by sacrificing a certain degree of process autonomy and incurring run-time and message overhead. Processes are required to coordinate their checkpointing actions in order to guarantee the consistency of corresponding checkpoints and hence the recovery line progression. In one experiment, it has been shown that the run-time overhead for coordinated checkpointing can be made reasonably small for a set of benchmark programs if optimization techniques primarily involving changes to the operating systems can be employed [12]. However, for many practical applications in which modifying the operating system is not considered a feasible solution, process autonomy in taking application-level checkpoints is essential for reducing run-time overhead by checkpointing when the process state is minimal.

*Log-based approach* [13–21] eliminates the domino effect by assuming *piecewise deterministic execution model* [22] which views process execution as consisting of a number of deterministic *state intervals*, each started by a nondeterministic event such as processing a new message. Nondeterministic event logging in addition to checkpointing is employed to reduce rollback propagation through deterministic state recon-

struction. However, it has been pointed out that the assumption of piecewise determinism may not be valid for the entire process execution, and hence the support for general nondeterministic execution is important [23].

This paper mainly considers uncoordinated checkpointing. The recovery line progression problem is addressed elsewhere [24]. Essentially, a domino-free unifying framework can be built by considering uncoordinated checkpointing as the basic scheme, and *checkpoint coordination* (whenever desirable) and *exploiting piecewise determinism* (whenever possible) as two mechanisms for bounding rollback propagation. More specifically, communication-induced checkpoints in a *lazy checkpoint coordination* scheme [24], and the *logical checkpoints* [25] obtained through event logging when piecewise determinism is available provide additional checkpoints to advance the recovery line.

The main focus of this paper is on the space overhead problem of uncoordinated checkpointing (and the unifying framework as well). Traditionally, garbage collection has been based on the notion of *obsolete* checkpoints and message logs: the global recovery line which suffices to recover from the failure of the entire system is computed, and all the obsolete checkpoints and message logs before that recovery line are no longer useful and can be discarded. In contrast, all the *non-obsolete* checkpoints and message logs have been assumed to be possibly useful for some future recovery and should be retained. With the possibility of domino effects, the space overhead may become prohibitively high.

Motivated by the observation that being obsolete is simply a sufficient condition for being garbage, we previously derived the necessary and sufficient condition for identifying all garbage checkpoints, which leads to an optimal checkpoint reclamation algorithm [26]. By using the approach of recovery line transformation and decomposition, we have demonstrated that any non-garbage checkpoint belonging to a possible future recovery line must also be contained in one of the $N$ "immediate future" recovery lines, where $N$ is the number of processes. In this paper, we apply the same approach to solving the problem of optimal message log reclamation[2] for systems requiring message logging to record *in-transit*, i.e., "sent but not yet received,"

[2] A simple *sufficient* condition based on *local* information has been presented to identify some garbage messages before they are logged [3]; this paper derives the *necessary and sufficient* condition based on *global* information for identifying all garbage message logs.

messages. We will show that any non-garbage message log which can become an in-transit message with respect to a possible future recovery must also be an in-transit message with respect to one of the $N$ "immediate future" recovery lines. We wish to stress that the message logs considered in this paper are used to record *the state of the channels* [8], and are different from the message logs used for deterministic replay in the log-based recovery schemes [13, 15]. While both *message contents* and *ordinal positions* are important in the latter, only message contents are needed in the former. In Section 5, we will also demonstrate the applicability of the optimal garbage collection algorithm to executions that exploit piecewise determinism.

The paper is organized as follows. Section 2 describes the checkpointing and recovery protocol; Section 3 derives the necessary and sufficient condition for identifying all garbage message logs, based on recovery line transformation and decomposition; experimental evaluation is described in Section 4; Section 5 extends our work to a partially-exploited piecewise deterministic model, and Section 6 concludes with a summary.

## 2 Checkpointing and Recovery Protocol

The system considered in this paper consists of a number of concurrent processes for which all process communication is through message passing. Processes are assumed to run on fail-stop processors [27] and, for the purpose of presentation, each process is considered an individual *recovery unit*. In order to allow general nondeterministic execution, we do not assume a piecewise deterministic model. This implies whenever the sender of a message $m$ rolls back and *unsends* $m$, the receiver which has already processed $m$ must also roll back to undo the effect of $m$ because the potential nondeterminism preceding the sending of $m$ (Fig. 1(a)) may prevent the same message from being resent during reexecution. Let $c_{i,x}$ denote the $x$th checkpoint ($x \geq 0$) of process $p_i$ ($0 \leq i \leq N-1$), where $N$ is the number of processes in the system. Two checkpoints $c_{i,x+1}$ and $c_{j,y}$ are then considered *inconsistent* if there is any message sent after $c_{j,y}$ and processed before $c_{i,x+1}$, i.e., $c_{j,y}$ *happened before* $c_{i,x+1}$ [28], or vice versa. In contrast, when the receiver $p_i$ of a message $m'$ rolls back and *unreceives* $m'$ (Fig. 1(b)), the sender $p_j$ may *not* need to roll back to *unsend* $m'$. If the acknowledge message for every normal message is treated as an additional dependency-carrying message, such an *in-transit message* $m'$ may be retrieved through a reliable end-to-end transmission protocol

[17, 29]. Alternatively, message $m'$ can be retrieved from a synchronous [13, 14] or an asynchronous [3, 15] message log. Therefore, checkpoints $c_{i,x}$ and $c_{j,y}$ in Fig. 1(b) are considered consistent.

During normal execution, each process periodically and independently takes its checkpoints. The interval between $c_{i,x}$ and $c_{i,x+1}$ is called the $x$th *checkpoint interval* of $p_i$, denoted by $(i, x)$. Each message is tagged with the process number and the current checkpoint interval number of the sender, and each receiver $p_i$ performs *direct dependency tracking* [1, 30] as follows: if a message sent from $(j, y)$ is processed in $(i, x)$, the direct dependency of $c_{i,x+1}$ on $c_{j,y}$ is recorded.

A garbage collection procedure can be periodically invoked by any process $p_i$. First, $p_i$ collects the direct dependency information from all the other processes to construct the *checkpoint graph* [1] as shown in Fig. 2(b). Then the *rollback propagation algorithm* (Fig. 3) is applied to the checkpoint graph to determine the *global recovery line*[3] (black vertices in Fig. 2(b)), before which all the checkpoints and message logs are obsolete and can be discarded. When any process initiates a rollback, it starts a similar procedure for recovery. The current volatile states of the surviving processes are treated as additional *virtual checkpoints* [2] for constructing an *extended checkpoint graph* of which the recovery line is called the *local recovery line* (shaded vertices) and indicates the consistent state for the system to roll back to.

## 3 Optimal Message Log Reclamation

Since the purpose of message logging is to record in-transit messages needed for rollback recovery, a message log is *non-garbage* if and only if it can become *an in-transit message with respect to* a possible future recovery line or, for short, *intersect*[4] a possible future recovery line. We model a process execution as consisting of a number of *operational sessions* [2] and *recovery sessions*, where an operational session is the interval between the start of normal execution and the instance of rollback initiation, and between two consecutive operational sessions is a *recovery session*. Since a future process execution may contain any

[3]The global recovery line is used when the entire system fails; a local recovery line is used when only a subset of processes becomes faulty.

[4]Any message can only *intersect* a recovery line from the left to the right, as shown in Fig. 1(b), because intersecting in the other direction contradicts the fact that a recovery line cannot contain any inconsistent checkpoints.

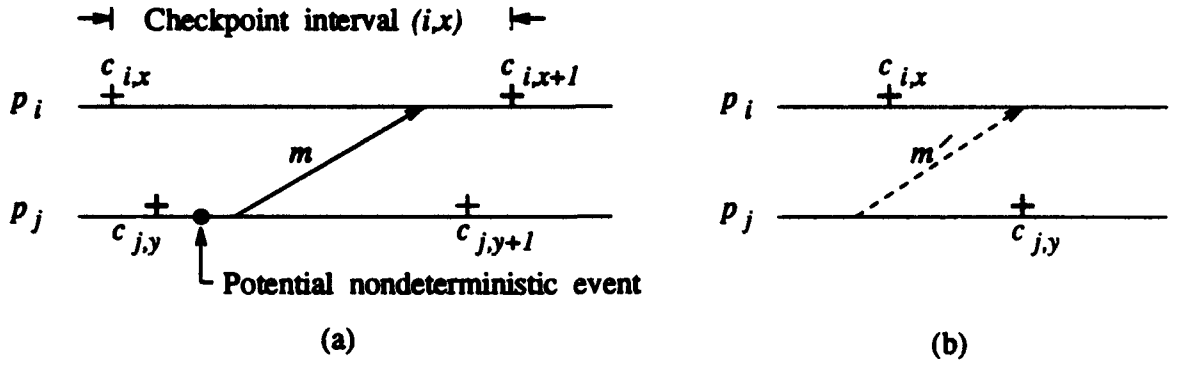



Figure 1: Checkpoint consistency. (a) inconsistent checkpoints $c_{j,y}$ and $c_{i,x+1}$; (b) in-transit message $m'$ and consistent checkpoints $c_{j,y}$ and $c_{i,x}$.

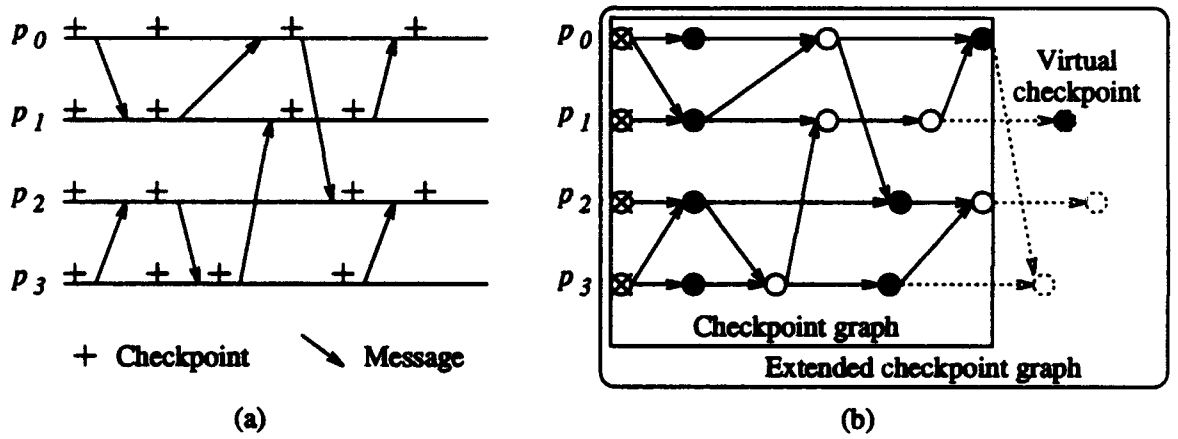



Figure 2: Checkpointing and rollback recovery. (a) example checkpoint and communication pattern; (b) checkpoint graph and extended checkpoint graph when $p_0$ initiates a rollback.

```
/* CP stands for checkpoint. Initially, all the CPs are unmarked */
include the latest CP of each process in the root set;
mark all CPs strictly reachable [31] from any CP in the root set;
while (at least one CP in the root set is marked) {
   replace each marked CP in the root set by the latest unmarked CP on the same process;
   mark all CPs strictly reachable from any CP in the root set;
}
the root set is the recovery line.
```

Figure 3: The rollback propagation algorithm.

number of arbitrary operational sessions and recovery sessions, there are an infinite number of possible future recovery lines. We first describe the recovery line transformation and decomposition which can provide a finite set of "immediate future" recovery lines sufficient for representing the infinite future possibilities for the purpose of garbage collection.

### 3.1 Recovery Line Transformation and Decomposition

Based on the previous description of checkpoint consistency, we define a *consistent global checkpoint* as a set of $N$ checkpoints, one from each process and no two of which are related through the *happened before* relation. A *recovery line* refers to the "latest available" consistent global checkpoint.

Since a checkpoint graph represents program dependency, vertices must be added to and removed from the graph according to a specific set of rules. In an operational session, new vertices are added to the checkpoint graph and can not have any outgoing edges to any existing vertices[5]. (If a graph $G'$ can be obtained by adding new vertices to another graph $G$ in this way, $G'$ is called a *potential supergraph* of $G$.) In a recovery session, existing vertices after the local recovery line are removed together from the checkpoint graph. The above rules for checkpoint graph evolution then determine the possible future checkpoint graphs, and therefore the possible future recovery lines.

We first define a set of $2^N$ *immediate supergraphs* which are the supergraphs of $G$ and the subgraphs of $\hat{G}$ as shown in Fig. 4. $\hat{G}$ is constructed by adding a *new-node* $n_i$ with single incoming edge at the end for each process $p_i$. Let $U$ denote the set of all such *new-nodes* and $\mathcal{RL}(G)$ denote the recovery line of a checkpoint graph $G$. Given any possible future recovery line $\mathcal{RL}(G')$ of $G$, we first apply *recovery line transformation* to $\mathcal{RL}(G')$ to transform it backwards in time into one of the $2^N$ recovery lines $\mathcal{RL}(\hat{G} - W), W \subseteq U$, followed by a *recovery line decomposition* to express the latter recovery line in terms of the $N$ recovery lines $\mathcal{RL}(\hat{G} - n_i), 0 \leq i \leq N-1$. We will demonstrate that since recovery line transformation and decomposition preserve all non-garbage checkpoints and message logs of any possible future recovery line, the above $N$ recovery lines suffice for the purpose of optimal garbage collection.

The example shown in Fig. 5 will be used to illustrate the transformation and decomposition throughout

[5]Vertices with incoming edges from not-yet-collected vertices are temporarily excluded from the checkpoint graph.

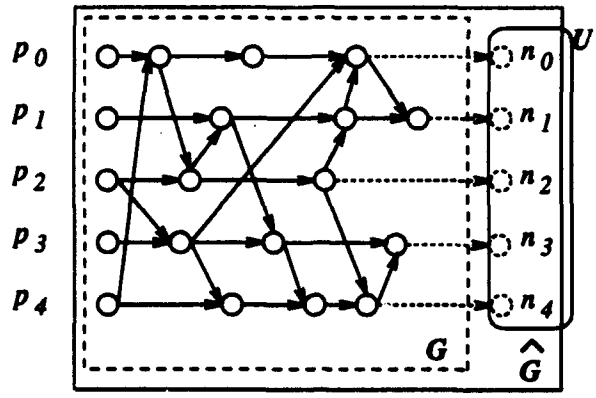



Figure 4: The immediate supergraphs.

this paper. (Formal proofs can be found elsewhere [26].) Suppose $G$ in Fig. 5(a) is the current checkpoint graph considered for garbage collection. Fig. 5(b) shows the extended checkpoint graph when $p_3$ later initiates the first rollback, and $G_c$ is the checkpoint graph immediately after the recovery. Fig. 5(d) shows another possible extended checkpoint graph when $p_0$ initiates a second rollback. We now describe how to transform and decompose $\mathcal{RL}(G_d)$, a possible future recovery line of $G$.

**Recovery Line Transformation:** Since any future process execution can be viewed as consisting of a number of operational sessions and recovery sessions, our approach is to define two elementary transformations: *transformation within an operational session* and *transformation across a recovery session.* Any possible future recovery line can then be transformed by repeatedly and alternately applying the two transformations.

**Transformation within an operational session:** First we consider $G_c$ and $G_d$, where $G_c$ is at the beginning of of a new operational session and $G_d$ is a potential supergraph of $G_c$. Given the recovery line $\mathcal{RL}(G_d)$, we replace the checkpoints $X$, $Y$ and $Z$ *which are not in* $G_c$ with their corresponding *new-nodes* $P$, $Q$ and $R$ of $G_c$, as shown in Fig. 5(g). $\mathcal{RL}(G_d) = \{A, B, X, Y, Z\}$ is then transformed into $\mathcal{RL}(G_g) = \{A, B, P, Q, R\}$, where $G_g$ is an immediate supergraph of $G_c$.

**Transformation across a recovery session:** We next consider $G_g$ and $G_b$ which is the checkpoint graph at the end of the first operational session (without the virtual checkpoints). Given the recovery line $\mathcal{RL}(G_g)$, we replace the two *new-nodes* $Q$ and $R$ *which are contributed by the rolled-back processes in the first recovery* with their corresponding checkpoints on the local recovery line, namely, $C$ and $D$. $\mathcal{RL}(G_g)$

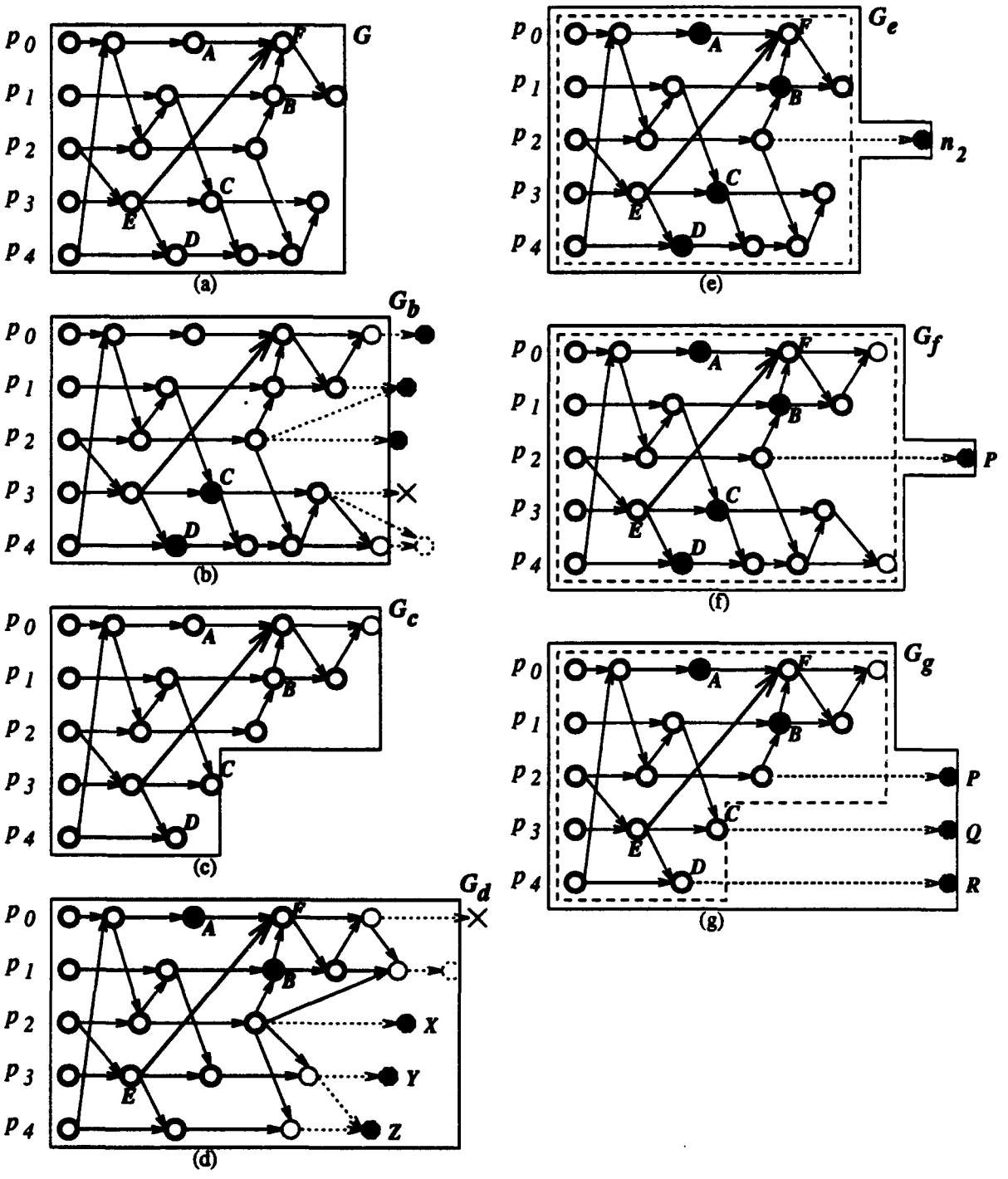



Figure 5: Example recovery line transformation.

is then transformed into $\mathcal{RL}(G_f) = \{A, B, P, C, D\}$, where $G_f$ is recognized as an immediate supergraph of $G_b$.

Finally, since $G_f$ is a potential supergraph of $G$, $\mathcal{RL}(G_f)$ can be transformed into $\mathcal{RL}(G_e) = \{A, B, n_2, C, D\}$. The possible future recovery line $\mathcal{RL}(G_d)$ is then transformed into the recovery line of an immediate supergraph $G_e$ of $G$.

**Recovery Line Decomposition:** Let $min(S)$ denote the set of *minimal elements*, i.e., vertices without any incoming edges, of $S$. The recovery line decomposition expresses each of the $2^N$ recovery lines in terms of the $N$ recovery lines $\mathcal{RL}(\hat{G} - n_i), 0 \leq i \leq N-1$, as follows.

$$\mathcal{RL}(\hat{G} - W) = min(\bigcup_{n_i \in W} \mathcal{RL}(\hat{G} - n_i)). \tag{1}$$

For example, the recovery line of $G_e = \hat{G} - \{n_0, n_1, n_3, n_4\}$ in Fig. 5(e) has the following decomposition (referring to Fig. 6):

$$\begin{aligned} \mathcal{RL}(G_e) &= min(\mathcal{RL}(\hat{G} - n_0) \cup \mathcal{RL}(\hat{G} - n_1) \cup \mathcal{RL}(\hat{G} - n_3) \cup \mathcal{RL}(\hat{G} - n_4)) \\ &= min(\{A, B, n_2, n_3, n_4, n_0, I, n_1, J, C, D\}) = \{A, B, n_2, C, D\}. \end{aligned}$$

### 3.2 Message Log Reclamation

Based on the approach of recovery line transformation and decomposition, it has been shown that the union of the $N$ recovery lines $\mathcal{RL}(\hat{G} - n_i), 0 \leq i \leq N-1$, contains all the non-garbage checkpoints [26]. For the example shown in Fig. 6, while all the checkpoints in $G$ are non-obsolete, only the shaded checkpoints in Fig. 6(f) are non-garbage and need to be retained. We next demonstrate that the set of in-transit messages with respect to the $N$ recovery lines contains all the non-garbage messages.

Instead of considering each individual message, we use its corresponding edge in the checkpoint graph for our discussion. Let $(a, b)$ denote the directed edge from vertex $a$ to vertex $b$. By definition, $(a, b)$ *intersects* a recovery line $\mathcal{RL}(G)$ if $a$ is on the left hand side of $\mathcal{RL}(G)$ and $b$ is on the right hand side of $\mathcal{RL}(G)$.

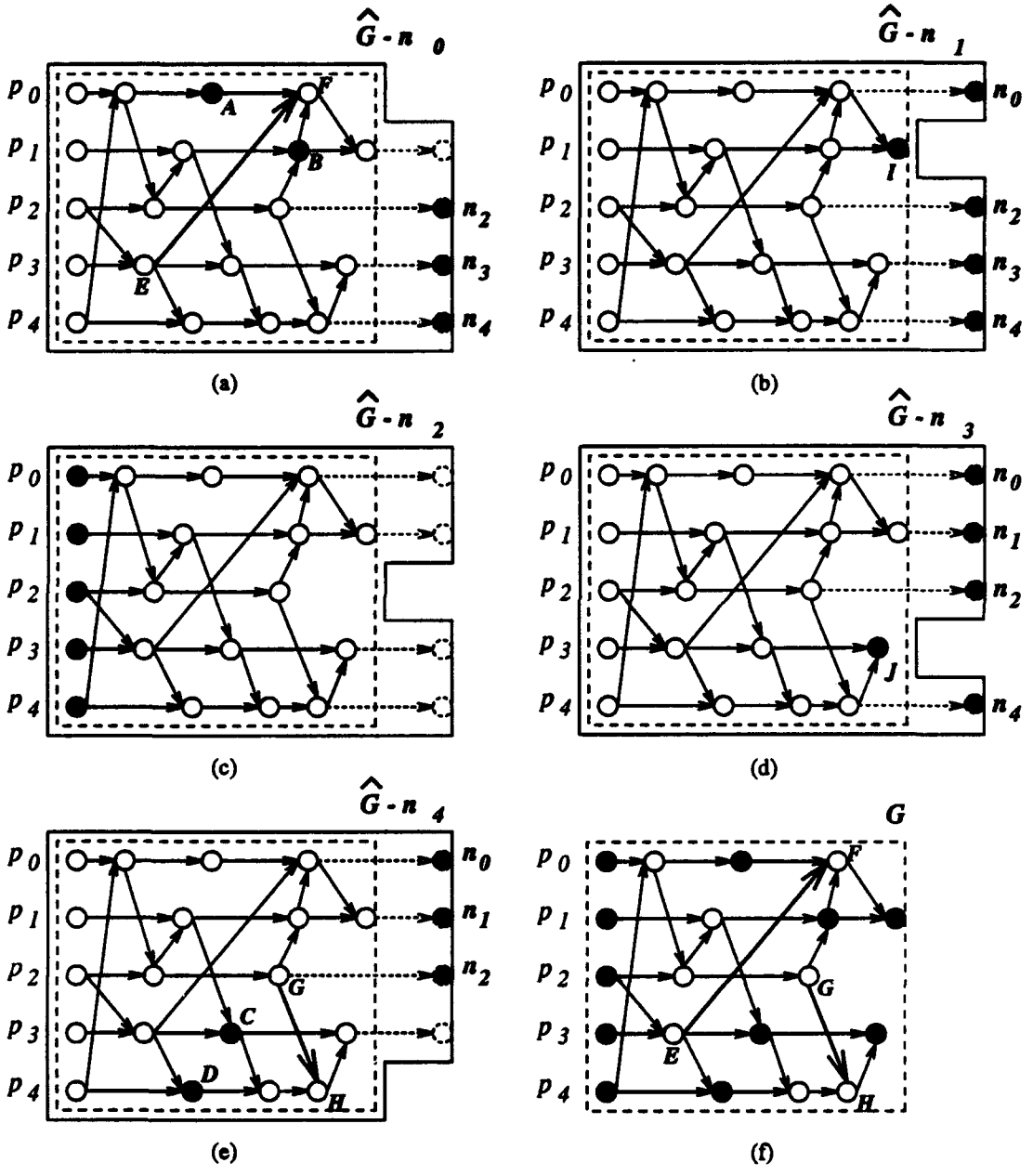



Figure 6: Example execution of our algorithm.

**PROPERTY 1** *Given a checkpoint graph $G$ and one of its edges $(a, b)$, if $(a, b)$ intersects a possible future recovery line, $(a, b)$ must intersect $\mathcal{RL}(\hat{G} - W)$ for some $W \subseteq U$.*

*Sketch of the proof.* Again, we use the example in Fig. 5. In particular, since the edge $(E, F)$ in $G$ *intersects* a possible future recovery line $\mathcal{RL}(G_d)$, we want to show that $(E, F)$ must also *intersect* $\mathcal{RL}(G_e)$ where $G_e = \hat{G} - W, W = \{n_0, n_1, n_3, n_4\}$.

**Transformation within an operational session:** First, we consider the relative position of any remaining checkpoint of $G$ to the recovery lines $\mathcal{RL}(G_d)$ and $\mathcal{RL}(G_g)$. Any such checkpoint which is on the left (right) hand side of $\mathcal{RL}(G_d)$ must remain on the left (right) hand side of $\mathcal{RL}(G_g)$. Therefore, any edge of $G$ *intersecting* $\mathcal{RL}(G_d)$, for example $(E, F)$, must also *intersect* $\mathcal{RL}(G_g)$ after the recovery line transformation.

**Transformation across a recovery session:** We next consider $\mathcal{RL}(G_g)$ and $\mathcal{RL}(G_f)$. Any remaining vertex of $G$ which is on the right hand side of $\mathcal{RL}(G_g)$ must remain on the right hand side of $\mathcal{RL}(G_f)$; those on the left hand side of $\mathcal{RL}(G_g)$ remain on the left hand side of $\mathcal{RL}(G_f)$ except for $C$ and $D$. Therefore, any remaining edge $(a, b)$ of $G$ *intersecting* $\mathcal{RL}(G_g)$ must also *intersect* $\mathcal{RL}(G_f)$ except for the possible outgoing edges of $C$ and $D$. But since $C$ and $D$ are on the local recovery line, all their outgoing edges must have been removed during the rollback; so any such $(a, b)$ must also *intersect* $\mathcal{RL}(G_f)$. Again, $(E, F)$ serves as such an example.

Finally, we can show that $(E, F)$ also *intersects* $\mathcal{RL}(G_e)$ by again applying the transformation within an operational session. □

Before applying the recovery line decomposition, we first express Eq. (1) in another form which is more convenient for considering the relative position of a checkpoint to a recovery line.

**LEMMA 1** *$min(\bigcup_{n_i \in W} \mathcal{RL}(\hat{G} - n_i))$ in Eq. (1) consists of the leftmost checkpoint of each process in the union.*

*Proof.* If a checkpoint $v$ of $p_i$ is not the leftmost checkpoint of $p_i$ in the union, then $v$ can not be a minimal element because there exists at least one checkpoint on its left. Conversely, if $v$ is the leftmost

checkpoint of $p_i$, $v$ must be in $min(\bigcup_{n_i \in W} \mathcal{RL}(\hat{G} - n_i))$ because there are only $N$ leftmost checkpoints and $\mathcal{RL}(\hat{G} - W) = min(\bigcup_{n_i \in W} \mathcal{RL}(\hat{G} - n_i))$ must contain $N$ checkpoints. □

**PROPERTY 2** *Given a checkpoint graph $G$ and one of its edges $(a, b)$, if $(a, b)$ intersects $\mathcal{RL}(\hat{G} - W)$ for some $W \subseteq U$, $(a, b)$ must intersect $\mathcal{RL}(\hat{G} - n_i)$ for some $0 \leq i \leq N - 1$.*

*Proof.* We will prove the property by showing that if $(a, b)$ does not *intersect* any $\mathcal{RL}(\hat{G} - n_i)$, $(a, b)$ cannot *intersect* any $\mathcal{RL}(\hat{G} - W)$.

Suppose $(a, b)$ does not *intersect* any $\mathcal{RL}(\hat{G} - n_i)$. Then, each $\mathcal{RL}(\hat{G} - n_i)$ must lie either entirely on the right hand side of $(a, b)$ or entirely on the left hand side of $(a, b)$.

**Recovery line decomposition:** Given any $\mathcal{RL}(\hat{G} - W)$, $W \subseteq U$, if all $\mathcal{RL}(\hat{G} - n_i)$'s, $n_i \in W$, are entirely on the right hand side of $(a, b)$, $\mathcal{RL}(\hat{G} - W)$ must also lie on the right hand side of $(a, b)$ by Eq. (1) and Lemma 1; if at least one $\mathcal{RL}(\hat{G} - n_i)$, $n_i \in W$, lies entirely on the left hand side of $(a, b)$, $\mathcal{RL}(\hat{G} - W)$ will be on the left hand side of $(a, b)$ again by Lemma 1. In either case, $(a, b)$ cannot *intersect* $\mathcal{RL}(\hat{G} - W)$. □

We are now prepared to prove the major result of this paper: the necessary and sufficient condition for a message log to be non-garbage.

**THEOREM 1** *A message log with its corresponding edge contained in $G$ is non-garbage if and only if the edge intersects $\mathcal{RL}(\hat{G} - n_i)$ for some $0 \leq i \leq N - 1$.*

*Proof.* Any non-garbage message log must have its corresponding edge $(a, b)$ *intersecting* a possible future recovery line. From Property 1, $(a, b)$ must *intersect* $\mathcal{RL}(\hat{G} - W)$ for some $W \subseteq U$. From Property 2, $(a, b)$ must *intersect* $\mathcal{RL}(\hat{G} - n_i)$ for some $0 \leq i \leq N - 1$. The *if* part comes from the fact that every $\mathcal{RL}(\hat{G} - n_i)$ is a possible future recovery line. □

Theorem 1 also leads an optimal message log reclamation algorithm for finding all non-garbage message logs: first compute the $N$ recovery lines $\mathcal{RL}(\hat{G} - n_i)$, $0 \leq i \leq N - 1$; only those message logs with their corresponding edges *intersecting* any of the $N$ recovery lines are non-garbage. In Fig. 6, the edge $(E, F)$

*intersects* $\mathcal{RL}(\hat{G} - n_0)$, $(G, H)$ *intersects* $\mathcal{RL}(\hat{G} - n_4)$ and none of the edges *intersects* $\mathcal{RL}(\hat{G} - n_1)$, $\mathcal{RL}(\hat{G} - n_2)$ or $\mathcal{RL}(\hat{G} - n_3)$. Therefore, while all the edges in Fig. 6(f) are non-obsolete, only those message logs corresponding to $(E, F)$ and $(G, H)$ need to be retained.

The complexity of the algorithm is analyzed as follows. The rollback propagation algorithm in Fig. 3 is of complexity $\mathbf{O}(|E|)$, where $|E|$ is the number of edges, because every edge marked by the algorithm can be removed. The remaining incoming edges of those checkpoints on the right hand side of the recovery line then give the set of edges *intersecting* the recovery line. Since the complexity of scanning through the above set of checkpoints is no greater than $\mathbf{O}(|E|)$, the complexity remains $\mathbf{O}(|E|)$. Our optimal garbage collection algorithm involves executing the rollback propagation algorithm on $N$ checkpoint graphs and is therefore of complexity $\mathbf{O}(N|E|)$.

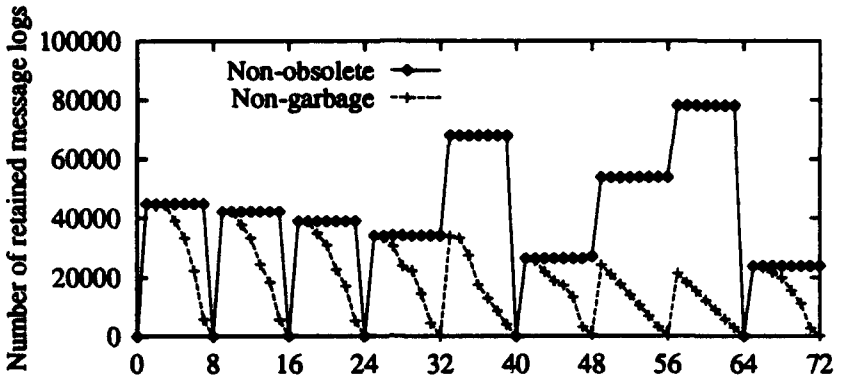
# 4 Experimental Evaluation

Three hypercube programs are used to illustrate the message log reclamation capabilities and benefits of our algorithm. They are Cell placement, Channel router and QR decomposition, running on an 8-node Intel iPSC/2 hypercube. Communication traces are collected by intercepting the `send` and `receive` system calls. Communication trace-driven simulation is then performed to obtain the results. The execution time for each program is listed in Table 1. The checkpoint interval is chosen to be approximately one tenth of the execution time.

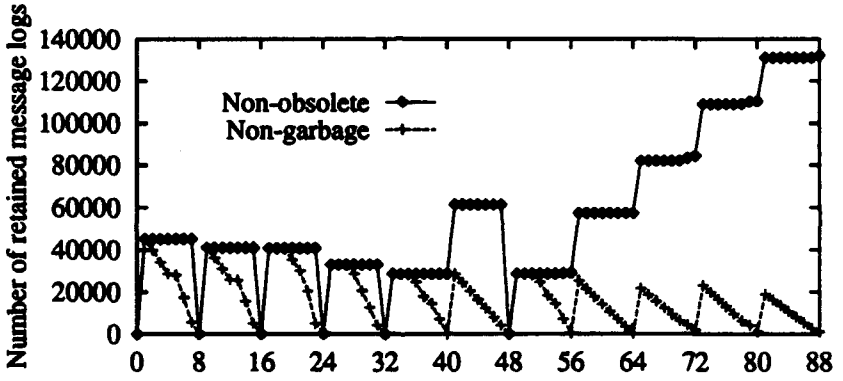
Table 1: Execution time and checkpoint interval.

| Programs | Cell placement | Channel router | QR decomposition |
|---|---|---|---|
| Execution time (sec) | 324 | 469 | 370 |
| Checkpoint interval (sec) | 35 | 40 | 35 |

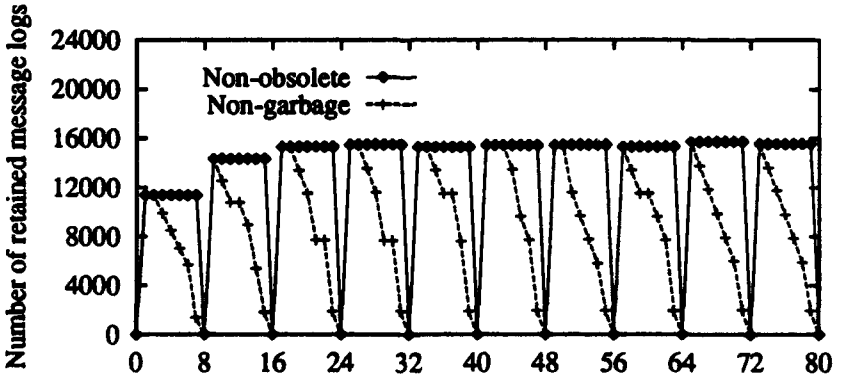
Figs. 7 compares our algorithm with the traditional algorithm for the three programs in terms of the number of retained message logs. Each curve shows the remaining space overhead after garbage collection if the algorithm is invoked after a certain number of checkpoints have been taken. Since the checkpointing




(a)




(b)




(b)

Figure 7: Message log reclamation results for the three parallel programs.

clocks on all nodes are approximately synchronized, checkpoints #8$n$ through #8($n$+1)-1 are taken at about the same time, which explains the fact that the number of messages is almost constant within that interval.

The domino effect is illustrated by the constant increase in the number of non-obsolete message logs as the total number of checkpoints increases, for example, between checkpoints #40 and #64 in Fig. 7(a) and between checkpoints #48 and #88 in Fig. 7(b). The figure shows that our algorithm performs consistently better than the traditional algorithm.

# 5 Exploiting Piecewise Determinism

Instead of viewing the piecewise deterministic (PWD) model as a constraint imposed upon the program behavior, we consider *exploiting piecewise determinism*, whenever possible and desirable, as a mechanism for bounding rollback propagation in an uncoordinated checkpointing protocol. The notion of *logical checkpoints* [25] has been proposed to provide a unified dependency model for both PWD and non-PWD scenarios. Essentially, referring to Fig. 8(b), the *physical checkpoint* $c_0$, the message log of $m_0$ (including both the message content and ordinal position) and the underlying PWD model equivalently place a logical checkpoint $L_0$ at the end of the state interval initiated by $m_0$ because of the capability of deterministic state reconstruction up to that point. Fig. 8(b) shows a situation where the PWD model is valid throughout the execution and so message logging can always be employed to insert additional logical checkpoints to effectively advance the recovery line, as compared with (a).

In practice, the PWD model may not be valid throughout the entire program execution; for example, it may not be appropriate to "replay" input events such as real-time clock readings and resource status. When the PWD model becomes invalid, it can only be resumed after the next physical checkpoint is taken because the deterministic state reconstruction for current checkpoint interval has been interrupted. Fig. 8(c) illustrates a situation where the PWD model can only be partially exploited. Suppose the piecewise determinism is not available for the parts of execution indicated by the shaded bars. Then the logical checkpoints belonging to those regions are no longer available. Fig. 9(a) shows the corresponding checkpoint graphs including all the physical and logical checkpoints; Fig. 9(b)-(f) apply the optimal garbage collection algorithm to the above

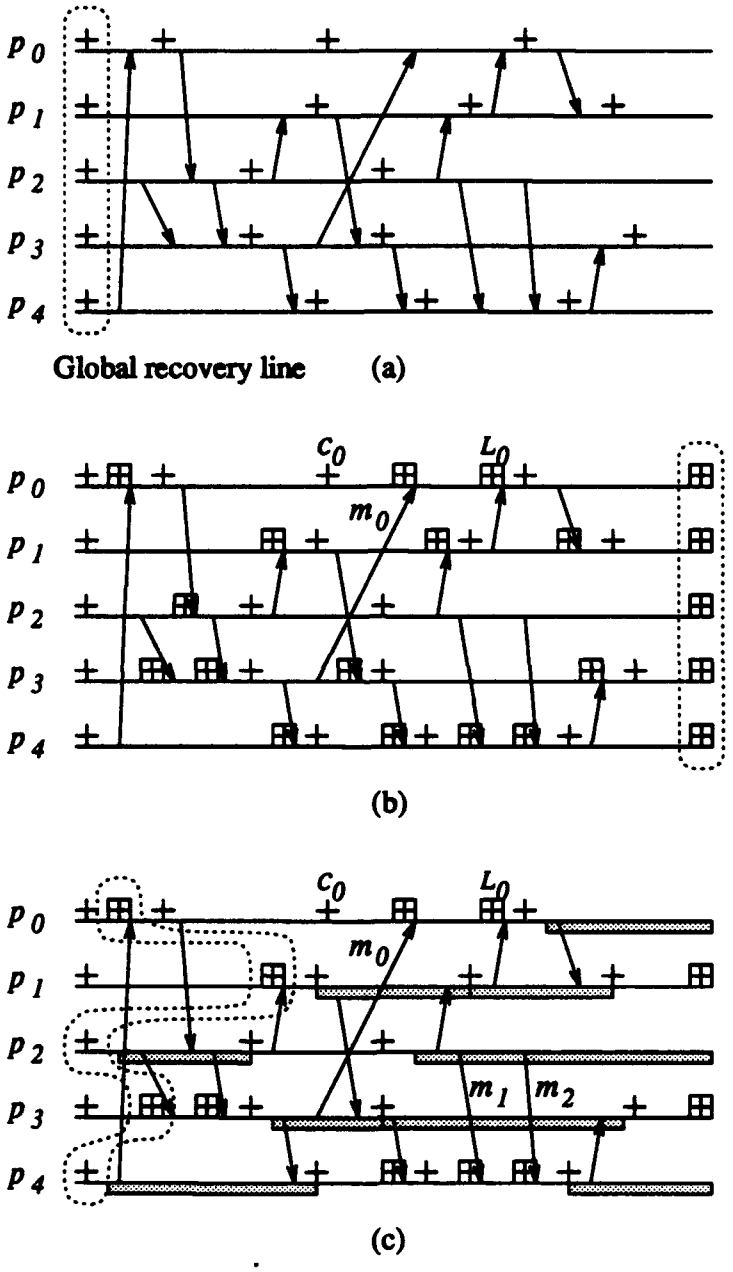



Figure 8: Piecewise determinism and the availability of logical checkpoints. (The shaded bars in (c) indicate those parts of process execution which do not satisfy the PWD model.)

checkpoint graph. Note that the logical checkpoint $L_0$ in Fig. 9(b) being non-garbage implies that both the physical checkpoint $c_0$ and the message log of $m_0$ (Fig. 8(c)) are non-garbage, and $m_0$ must be the first new message to be processed after $p_0$ restarts from $c_0$. In contrast, Fig. 9(f) identifies the two thick edges as non-garbage edges which means the contents of the message logs of $m_1$ and $m_2$ (Fig. 8(c)) are non-garbage but the ordinal position information can be discarded [6].

# 6 Summary

For systems requiring message logging to record in-transit messages, we have derived the necessary and sufficient condition for identifying all garbage message logs, which leads to an optimal message log reclamation algorithm. Combining it with a previous optimal checkpoint reclamation algorithm, we have developed an optimal garbage collection algorithm for minimizing the space overhead of uncoordinated checkpointing. The overall complexity of the algorithm is $O(N|E|)$ where $N$ is the number of processes and $|E|$ is the number of edges in the checkpoint graph. Communication trace-driven simulation results for three parallel programs showed that the algorithm can be effective in reducing the space overhead for real applications.

# Acknowledgement

The authors wish to express their sincere thanks to Andy Lowry (IBM) and Pi-Yu Chung (Illinois) for their valuable discussions, to Junsheng Long (North Carolina) for his help with the experimental results and to Prith Banerjee (Illinois) for his hypercube programs.

# References

[1] K. Tsuruoka, A. Kaneko, and Y. Nishihara, "Dynamic recovery schemes for distributed processes," in *Proc. IEEE 2nd Symp. on Reliability in Distributed Software and Database Systems*, pp. 124–130, 1981.

[6] This does not include the possible FIFO order for messages in the same channel.

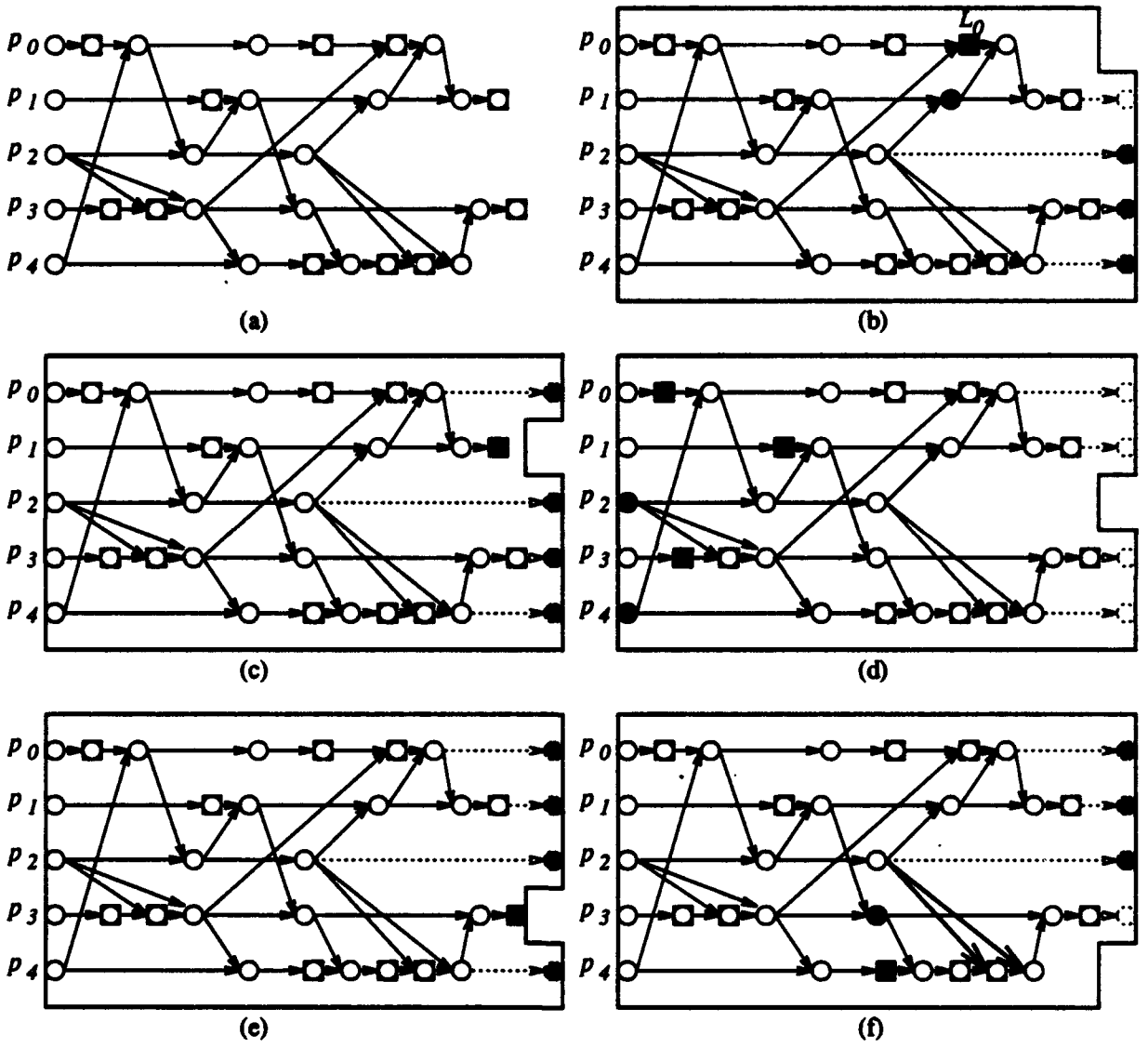



**Figure 9: Checkpoint graphs and optimal garbage collection for a partially exploited piecewise deterministic model.**

[2] B. Bhargava and S. R. Lian, "Independent checkpointing and concurrent rollback for recovery - An optimistic approach," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 3–12, 1988.

[3] Y. M. Wang and W. K. Fuchs, "Optimistic message logging for independent checkpointing in message-passing systems," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 147–154, Oct. 1992.

[4] B. Randell, "System structure for software fault tolerance," *IEEE Trans. on Software Engineering*, vol. SE-1, pp. 220–232, June 1975.

[5] D. L. Russel, "State restoration in systems of communicating processes," *IEEE Trans. on Software Engineering*, vol. SE-6, pp. 183–194, Mar. 1980.

[6] Y. Tamir and C. H. Sequin, "Error recovery in multicomputers using global checkpoints," in *Proc. Int'l Conf. on Parallel Processing*, pp. 32–41, 1984.

[7] K. G. Shin and Y.-H. Lee, "Evaluation of error recovery blocks used for cooperating processes," *IEEE Trans. on Software Engineering*, vol. 10, no. 6, pp. 692–700, 1984.

[8] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. on Computer Systems*, vol. 3, pp. 63–75, Feb. 1985.

[9] K. Li, J. F. Naughton, and J. S. Plank, "Checkpointing multicomputer applications," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 2–11, 1991.

[10] T. H. Lai and T. H. Yang, "On distributed snapshots," *Information Processing Letters*, vol. 25, pp. 153–158, May 1987.

[11] D. Briatico, A. Ciuffoletti, and L. Simoncini, "A distributed domino-effect free recovery algorithm," in *Proc. IEEE 4th Symp. on Reliability in Distributed Software and Database Systems*, pp. 207–215, 1984.

[12] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel, "The performance of consistent checkpointing," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 39–47, Oct. 1992.

[13] A. Borg, J. Baumbach, and S. Glazer, "A message system supporting fault-tolerance," in *Proc. 9th ACM Symp. on Operating Systems Principles*, pp. 90–99, 1983.

[14] M. L. Powell and D. L. Presotto, "Publishing: A reliable broadcast communication mechanism," in *Proc. 9th ACM Symp. on Operating Systems Principles*, pp. 100–109, 1983.

[15] R. E. Strom and S. Yemini, "Optimistic recovery in distributed systems," *ACM Trans. on Computer Systems*, vol. 3, pp. 204–226, Aug. 1985.

[16] A. P. Sistla and J. L. Welch, "Efficient distributed recovery using message logging," in *Proc. 8th ACM Symposium on Principles of Distributed Computing*, pp. 223–238, 1989.

[17] D. B. Johnson and W. Zwaenepoel, "Recovery in distributed systems using optimistic message logging and checkpointing," *J. of Algorithms*, vol. 11, pp. 462–491, 1990.

[18] T. T.-Y. Juang and S. Venkatesan, "Crash recovery with little overhead," in *Proc. IEEE Int'l Conf. on Distributed Computing Systems*, pp. 454–461, 1991.

[19] A. Lowry, J. R. Russell, and A. P. Goldberg, "Optimistic failure recovery for very large networks," in *Proc. IEEE Symp. on Reliable Distr. Syst.*, pp. 66–75, 1991.

[20] E. N. Elnozahy and W. Zwaenepoel, "Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit," *IEEE Trans. on Computers*, vol. 41, pp. 526–531, May 1992.

[21] B. H. L. Alvisi and K. Marzullo, "Nonblocking and orphan-free message logging protocols," Tech. Rep. TR92-1317, Dept. of Computer Science, Cornell University, Dec. 1992.

[22] R. E. Strom, D. F. Bacon, and S. A. Yemini, "Volatile logging in n-fault-tolerant distributed systems," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 44–49, 1988.

[23] D. B. Johnson and W. Zwaenepoel, "Transparent optimistic rollback recovery," *ACM Operating Systems Review*, pp. 99–102, Apr. 1991.

[24] Y. M. Wang and W. K. Fuchs, "Lazy checkpoint coordination for bounding rollback propagation." To appear in *Prof. 12th Symp. on Reliable Distributed Systems*, Oct. 1993.

[25] Y. M. Wang, Y. Huang, and W. K. Fuchs, "Progressive retry for software error recovery in distributed systems," in *Proc. IEEE Fault-Tolerant Computing Symposium*, pp. 138–144, June 1993.

[26] Y. M. Wang, P. Y. Chung, I. J. Lin, and W. K. Fuchs, "Checkpoint space reclamation for independent checkpointing in message-passing systems," Tech. Rep. CRHC-92-06, Coordinated Science Laboratory, University of Illinois at Urbana–Champaign, 1992.

[27] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems," *ACM Trans. on Computer Systems*, vol. 1, pp. 222–238, Aug. 1983.

[28] L. Lamport, "Time, clocks and the ordering of events in a distributed system," *Comm. of the ACM*, vol. 21, pp. 558–565, July 1978.

[29] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. on Software Engineering*, vol. SE-13, pp. 23–31, Jan. 1987.

[30] Y. M. Wang, A. Lowry, and W. K. Fuchs, "Consistent global checkpoints based on direct dependency tracking." Research Report RC 18465, IBM T.J. Watson Research Center, Yorktown Heights, New York, Oct. 1992.

[31] K. P. Bogart, *Introductory combinatorics*. Pitman Publishing Inc., Massachusetts, 1983.